Fast and Optimized Task Allocation Method for Low Vertical Link Density 3-Dimensional Networks-on-Chip based Many Core Systems

Haoyuan Ying*, Thomas Hollstein† and Klaus Hofmann*
*Integrated Electronic Systems Lab, TU Darmstadt, Germany
Email: Haoyuan.Ying, Klaus.Hofmann@ise.tu-darmstadt.de
†Tallinn University of Technology, Estonia
Email: thomas@pld.ttu.ee

***Abstract*—The advantages of moving from 2-Dimensional Networks-on-Chip (NoCs) to 3-Dimensional NoCs for any application must be justified by the improvements in performance, power, latency and the overall system costs, especially the cost of Through-Silicon-Via (TSV). The trade-off between the number of TSVs and the 3D NoCs system performance becomes one of the most critical design issues. In this paper, we present a fast and optimized task allocation method for low vertical link density (TSV number) 3D NoCs based many core systems, in comparison to the classic methods as Genetic Algorithm (GA) and Simulated Annealing (SA), our method can save quite a number of design time. We take several state-of-the-art benchmarks and the generic scalable pseudo application (GSPA) with different network scales to simulate the achieved design (by our method), in comparison to GA and SA methods achieved designs, our technique can achieve better performance and lower cost. All the experiments have been done in GSNOC framework (written in SystemC-RTL), which can achieve the cycle accuracy and good flexibility.**

I. Introduction

Moving to advanced technologies has increased transistor density critically, which allows the designers to implement more complex SoCs. Interconnection infrastructure became main bottleneck of utilizing the available transistors. NoCs may become the replacement of the conventional interconnects due to its scalability and higher bandwidth [1]. With the increased number of cores, NoCs based communication infrastructures will also reach the latency and power overhead bottlenecks. To solve this problem, 3D ICs have been targeted as one of the solutions. Many technologies like die-to-die bonding, die-to-wafer bonding, Wafer-to-wafer bonding have been investigated for 3D integration [2]. As an interconnect for 3D integration, TSV technologies have got the highest acceptance and they are reasonably understood on the aspects of thermal, performance and yield properties [3].

According to the 2009 ITRS roadmap, TSV diameter will be in the range from 1.5 μm to 1.0 μm between the year of 2009 and 2015 [4]. During the same period, the area of a 4-transistor logic gate is projected to reduce from 0.82 μm^2 to 0.20 μm^2. This leads to the area ratio between TSVs and logic gates in the range from 2.74 (= 2.25/0.82) to 5 (= 1.0/0.20). If we consider the pitch of a TSV (between 3 μm and 5 μm), TSV-to-gate-size ratio will increase further. Thus, minimizing the TSV number saves considerable active area, which can be utilized for transistors. The most critical challenge for 3D ICs to be used in main stream products is cost trade-off [5], so transition from 2D ICs to 3D ICs must be justified in terms of performance improvement, power consumption and area.

978-3-9815370-0-0 / DATE13 / © 2013 EDAA

The 3D NoCs system performance cannot be degraded because of the TSV number reduction. Therefore, in this paper, we develop one fast and optimized task allocation technique for low TSV number 3D NoCs based many core system. Our method can optimize the system performance against the TSVs number reduction. In comparison to the classic GA and SA based methods, our method can save task allocation computation time (linear to the network scale, overall 98% advantage for 8x8x4 3D NoCs). The simulation results have shown that the design (by our method) can achieve better performance and lower cost in comparison to the system designs which achieved by GA and SA based methods. According to the simulation results, the 50% vertical link density setup achieved the best design trade-off point.

This paper is organized as following: Section II will describe the state-of-the-art related works, we will present our technique in detail in Section III, in Section IV, the SA and GA methods that we used to compare will be introduced. The experiment setup and results will be presented and analyzed in Section V, and the Section VI is for the conclusion and future work.

II. Related Works

In [6], the authors presented a constructive heuristic IP placement method (CMAP) which can be separated to two parts, as LBMAP (Link-based) and SBMAP (Sort-based). The results had shown that the CMAP could achieve less communication cost and less design computation time. However, there are two points can be optimized, one is that they had not considered about the bandwidth constrains of the network links while the mapping, the other is that there is no consideration about the link direction in their model.

In [7], the authors presented a GA based NoC topology generation method. Based on their method, during the Mutation

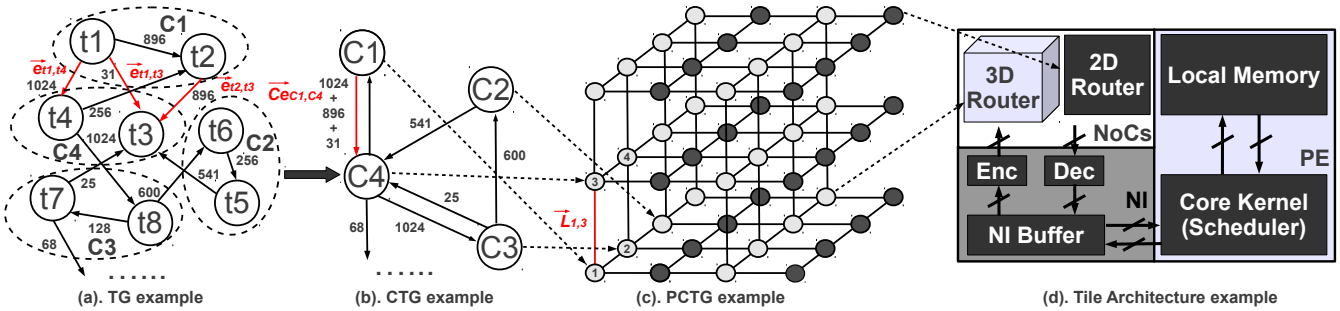

Fig. 1. TG, CTG and $PCTG$ examples

process, the maximal bandwidth capacities for all channels were considered not to be exceeded when assigning the network tile placements due to the communication requests. Once the task graph is quite complex and the timing analysis is incorrect, the method is hard to guarantee the optimization.

In [8], the authors also demonstrated GA algorithm based high-level 3D NoC synthesis technique. The key idea of this work is to find out the global minimal path for all the communications. The authors consider that the length of routing path strongly affects the power consumption, however, this can create more congestion nodes in the network, and the congestion can cause more power consumption and violate the system performance.

In [9], the authors presented a SA method to place the NoCs cores. The objective function they developed considered link bandwidth and latency. They achieved good results as distributing the traffic into the network as evenly as possible, and satisfied the link bandwidth constrains and reduced the latency.

In [10], the authors demonstrated a combined SAGA task allocation method. They used the GA at the start phase to explore a number of possible task allocation solutions, after several rounds, the best solution will be passed to the SA phase to achieve further optimization. The design computation time will be extremely long for very large systems in comparison to our method, and they only considered the overall execution time, some other important design factors as power, throughput and latency were missing.

In this paper, we will evaluate our method and the typical SA and GA methods for allocating tasks into 3D NoCs in terms of performance and cost.

III. Fast and Optimized Task Allocation Method

In this paper, we developed a hybridize constructive heuristic algorithm based (HyCH) fast and optimized task allocation method, some terminologies and constrains should be given in order to introduce our HyCH task allocation method.

$TG(t, \overrightarrow{e})$: Task graph, which connect a number of different tasks with a certain number of communication edges.

t: Task, which is indicated as the smallest executable unit in the entire application.

$\overrightarrow{e}$: The communication edge among tasks.

$w_{\overrightarrow{e}}$: The communication weight of the $\overrightarrow{e}$, it is counted as the number of flits.

Figure 1 (a) has shown the TG example. There are eight ts, and 12 $\overrightarrow{e}$s to connect all of them. The numbers around each $\overrightarrow{e}$ indicate its communication weight.

In some applications TGs, some ts are tightly connected, and it is inefficient to allocate the tightly connected ts into different processing elements (PE). Therefore, in this paper, we set some clusters to contain the tightly connected ts.

$CTG(C, \overrightarrow{Ce})$: Cluster task graph, which represents the task graph in a clustered view.

C: Task clusters, some tightly connected tasks should be allocate in the same task cluster.

$\overrightarrow{Ce}$: The communication edges among task clusters.

$w_{\overrightarrow{Ce}}$: The communication weight of the $\overrightarrow{Ce}$s. The $w_{\overrightarrow{Ce}}$ should include all the $w_{\overrightarrow{e}}$s from the same C to another.

n_C: The total number of the Cs in the CTG.

$n_{\overrightarrow{Ce}}$: The total number of the inter $\overrightarrow{Ce}$s in the CTG.

$bw_{\overrightarrow{Ce}}$: The bandwidth requirement of the $\overrightarrow{Ce}$.

Figure 1 (a) and (b) have shown the conversion from TG to CTG. The eight tasks have been merged into four clusters (C_1 to C_4). The inter cluster communication edges should take all the corresponding $\overrightarrow{e}$s into consideration. For example, the $\overrightarrow{Ce}_{C1,C4}$ contains communication information of the $\overrightarrow{e}_{t1,t3}$, $\overrightarrow{e}_{t1,t4}$ and $\overrightarrow{e}_{t2,t3}$, because the $t3$ and $t4$ are assigned to the same cluster (C_2). Also, the $w_{\overrightarrow{Ce}_{C1,C4}}$ equals to the sum of $w_{\overrightarrow{e}_{t1,t3}}$, $w_{\overrightarrow{e}_{t1,t4}}$ and $w_{\overrightarrow{e}_{t2,t3}}$.

Figure 1 (b) and (c) have shown the allocation process of Cs and the low vertical link density 3D NoCs topology. All the $\overrightarrow{Ce}$s are mapped to the tile to tile links. For each network tile, it should be a PE with either a 2D or 3D NoCs router according to the topology (Figure 1 (c) and (d)).

In this paper, we only focus on the allocation of different task clusters (Figure 1 (b) to (c)), the methodologies of clustering tasks are orthogonal to be considered here.

$PCTG(Tile, \overrightarrow{L}, VF)$: Physical Communication Topology Graph.

X, Y, Z: The total number of tiles in X, Y and Z dimension.

n_{Tile}: The total network tiles in the $PCTG$, n_{Tile} = X*Y*Z.

$Tile$: A single network tile in $PCTG$, which includes the processing element and communication infrastructure.

VF: Vertical link floor plan.

VD: Vertical link density.

$\overrightarrow{L}$: NoCs link, $\overrightarrow{L_{i,j}}$ indicates the link from $Tile_i$ to $Tile_j$.

$|\overrightarrow{L}|$: The Manhattan distance (counting as hops) of the the NoCs link $\overrightarrow{L}$.

$w_{\overrightarrow{L}}$: The communication weight of the link $\overrightarrow{L}$.

$bw_{\overrightarrow{L}}$: The bandwidth requirement of the the link $\overrightarrow{L}$.

M_{bw}: The physical bandwidth capacity of the NoCs links.

When our HyCH method is employed, (1) can represent the process:

$$HyCH(C_i) \mapsto Tile_j, \forall i \in [1, n_{C_e}], j \in [1, n_{Tile}]; \quad (1)$$

Assuming that

$$\begin{aligned} C_i \in Tile_k, C_j \in Tile_m, \forall i, j \in [1, n_C], k, m \in [1, n_{Tile}]; \\ \overrightarrow{Ce}_{i,j} \mapsto \overrightarrow{L}_{k,m}, \forall i, j \in [1, n_C], k, m \in [1, n_{Tile}]; => \\ w_{\overrightarrow{Ce}_{i,j}} = w_{\overrightarrow{L}_{k,m}}, \forall i, j \in [1, n_C], k, m \in [1, n_{Tile}]; \\ bw_{\overrightarrow{Ce}_{i,j}} = bw_{\overrightarrow{L}_{k,m}}, \forall i, j \in [1, n_C], k, m \in [1, n_{Tile}]; \end{aligned} \quad (2)$$

According to (2), after employing the HyCH method, all the Cs and $\overrightarrow{Ce}$s in the CTG are mapped on the $PCTG$ as $Tile$s and $\overrightarrow{L}$s, respectively (example has shown as Figure 1). In our HyCH method, we try to minimize the cost function (CF) as (3) represented, and this method can achieve better results than the GA and SA methods which are described in the Section IV can.

$$\begin{aligned} CF = \sum_{i,j=1}^{n_C} w_{\overrightarrow{Ce}_{i,j}} \times |\overrightarrow{Ce}_{i,j}| \times bw_{\overrightarrow{Ce}_{i,j}} = \\ \sum_{k,m=1}^{n_{Tile}} w_{\overrightarrow{L}_{k,m}} \times |\overrightarrow{L}_{k,m}| \times bw_{\overrightarrow{L}_{k,m}}, \\ \forall i, j \in [1, n_C], k, m \in [1, n_{Tile}]; \end{aligned} \quad (3)$$

In (3), if the accumulated $bw_{\overrightarrow{L}_{k,m}}$ is over the M_{bw}, the $bw_{\overrightarrow{L}_{k,m}}$ should be set as the M_{bw}.

Instead of allocating different tasks (or clusters in this paper) into different $Tile$s directly, the general idea of HyCH method is to map different $\overrightarrow{Ce}$s in the CTG on the $\overrightarrow{L}$s in the $PCTG$. The process of the HyCH method has two steps, the first step is to re-arrange all the $\overrightarrow{Ce}$s in the CTG and finally make an allocation order for them, the second step is to map all the re-arranged $\overrightarrow{Ce}$s on the $\overrightarrow{L}$s.

There are two phases to re-arrange all the $\overrightarrow{Ce}$s in the CTG to a better allocation order.

Phase 1. In this phase, the mapping order of the $\overrightarrow{Ce}$s will be re-arranged.

$Wtot_C$: The total communication weight which has the direct relation to the cluster C. For example, the $Wtot_{C_i}$ should include all communication data volume received by the cluster C_i and all the communication data volume injected by the C_i.

n_{dep}: The total number of $\overrightarrow{Ce}$s which are connected to the C. For example, in Figure 1, the $n_{dep_{C_4}}$ equals to six, because C_4 connects six $\overrightarrow{Ce}$s.

The order factor of Cs (O_C) will be calculated as:

$$O_{C_i} = \frac{Wtot_{C_i}}{n_{dep_{C_i}}}, i \in [1, n_C]; \quad (4)$$

According to (4), we re-arrange all the $\overrightarrow{Ce}$s in the order from the high O_{C_i} to the low ones which the $\overrightarrow{Ce}$ contains.

Phase 2. Although the higher dense Cs will be allocated first, there are still some potential optimization points. For example, if there is a $\overrightarrow{Ce}$ belongs to a high O_{C_i} C, however the $w_{\overrightarrow{Ce}}$ is very low, then this $\overrightarrow{Ce}$ can be mapped in a low priority and release its priority to other $\overrightarrow{Ce}$s which has low O_C but high $w_{\overrightarrow{Ce}}$. Therefore, in this phase, we maintain the order that set up by the phase 1, and slightly modify the order according to the $w_{\overrightarrow{Ce}}$s.

$w_{avg_{\overrightarrow{Ce}}}$: The average communication weight for all the $\overrightarrow{Ce}$s in the CTG.

$$w_{avg_{\overrightarrow{Ce}}} = \frac{\sum_{i,j=1}^{n_C} w_{\overrightarrow{Ce}_{i,j}}}{n_{\overrightarrow{Ce}}}; \quad (5)$$

By modifying the order phase 1 made, we set the $w_{avg_{\overrightarrow{Ce}}}$ as a threshold value, if the $w_{\overrightarrow{Ce}}$ is less than the threshold, it should be assigned a lower priority, and as versa.

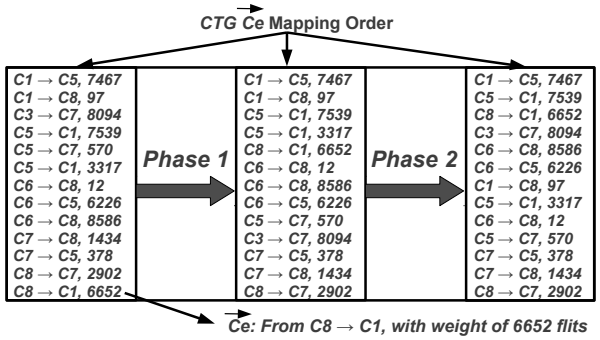

Fig. 2. Example of the Re-arrange Process (Phase 1 and 2)

Figure 2 has shown the example of the $\overrightarrow{Ce}$ re-arranging phase 1 and 2. After phase 1, the $\overrightarrow{Ce}$s are arranged according to the value of O_C, and after phase 2, the $\overrightarrow{Ce}$s order is re-arranged according to the previous order (phase 1) and the $w_{avg_{\overrightarrow{Ce}}}$ value. For example, the C_1 has the highest O_C, and at phase 1 the C_1 related $\overrightarrow{Ce}$s are arranged as the top priority to be mapped. The $\overrightarrow{Ce}$ $C_1 -> C_8$, 97 has the small volume of communication data, and it is should be assigned a lower mapping order at phase 2.

IV. GA and SA based Task Allocation Method

Many researchers have studied the SA and GA based task allocation methods for low power NoCs based many core

Algorithm 1 HyCH Task Allocation Algorithm Flow

HyCH starts:
Read Application (CTG) and NoCs Information;
Re-arrange the CTG $\overrightarrow{Ce}$ Phase 1;
Re-arrange the CTG $\overrightarrow{Ce}$ Phase 2;
while $m < n_{\overrightarrow{Ce}}$ **do**
  Assuming: $\overrightarrow{Ce}_i$ is the $\overrightarrow{Ce}$ from C_j to C_k;
  if C_j and C_k are neither allocated **then**
    Randomly Search the Free NoCs Position for C_j;
    Allocate C_k in another NoCs Position with the Least CF value;
  else if One of the C_j and C_k is not allocated **then**
    Allocate C_j (or C_k) in another NoCs Position with the Least CF value;
  else if C_j and C_k are both allocated **then**
    n = 0;
    while $n < n_C$ **do**
      if C_n is not allocated **then**
        Jump to Next $\overrightarrow{Ce}$;
      end if
      n++;
    end while
  end if
  Next $\overrightarrow{Ce}$: m++;
end while
End: Output the $PCTG$ **(With** CTG **Allocated)**;

system design [7] [8] [9] [10]. In this paper, we implement both the GA and SA according to several literatures, and compare the efficiently between the HyCH, GA and SA.

Cost Function

We used the well-known energy model as the cost function in GA and SA, since using this model is the most direct way to reduce the NoCs system power consumption.

$$E_{bit} = n_{router} \times E_{router_{bit}} + (n_{router} - 1) \times E_{wire_{bit}}; \quad (6)$$

(6) has shown the single bit communication energy consumption model for one communication edge ($\overrightarrow{Ce}$ in CTG). n_{router} is the number of hops that the $\overrightarrow{Ce}$ has to cross the network (the $|\overrightarrow{L}|$ after the mapping). Assuming that the channel width (flit bit-width) is n_w, and we already knew the communication weight of this $\overrightarrow{Ce}$ is $w_{\overrightarrow{Ce}}$, the overall communication energy can be modeled as:

$$E = \sum_{i,j=1}^{n_t} n_w \times w_{\overrightarrow{e}_{i,j}} \times E_{bit} = \sum_{k,m=1}^{n_{Tile}} n_w \times w_{\overrightarrow{L}_{k,m}} \times E_{bit} \quad (7)$$

For all of the parameters in (6), we extracted them from SYNOPSYS design compiler with TSMC 65 nm LP library.

GA Flow and Parameters:

$n_{population}$: The number of chromosomes in each generation. In this paper, we set the $n_{population}$ as 100.

n_{gen}: The total number of the generations in the GA algorithm. In this paper, we set the n_{gen} as 100.

n_{best}: The number of the chromosomes with better cost values can be fetched to next generation directly. In this paper, we set the n_{best} as 25.

$Mutation$: Mutation process in GA. This process will change the mapping of several Cs in the NoCs. In general, we simply exchange the positions of two Cs based on the randomly generated probabilities.

p_{ex}: The randomly generated position exchanging probabilities of the Cs.

p_{ex_T}: The position exchanging threshold of all the Cs, which is the same for all the Cs in the same generation, but it is randomly generated for each different generations.

$$Mutation(C_i) \mapsto Tile_{random().[1,n_{Tile}]}, \forall p_{ex_i} > p_{ex_T} \quad (8)$$

The certain new chromosomes are generated firstly by the crossover to fulfill the remained positions for the next generation (after passing n_{best} chromosomes directly from the current generation).

The crossover process has to select the parents first. The selection is based on the cost values of all the chromosomes. We fix a number of the chromosomes with the better cost values (in this paper, we fix the number as 50% of the population) in the current generation as the crossover parents candidates, and the crossover process will randomly select two candidates for one new chromosome.

Algorithm 2 GA Algorithm Flow

GA starts:
Read Application (CTG) and NoCs Information;
Load GA parameters;
Generate the First Generation ($n_{population}$);
while $n < n_{gen}$ **do**
  Calculate cost values of all the chromosomes;
  Fetch best n_{best} chromosomes directly to next generation;
  while $m < n_{population}$ - n_{best} **do**
    Crossover; Mutation; m++;
  end while
  Construct the next generation;
  m = 0; n++;
end while
End: Output the $PCTG$ **(With** CTG **Allocated)**;

SA Flow and Parameters:

$Temp$: Temperature. In this paper, we set the initial $Temp$ as 10000.

$ETemp$: Ending Temperature, once the temperature reaches this value, the SA should be finished. In this paper, we set $ETemp$ as 100.

CS: Cooling Step, which helps the temperature to be reduced, in the range between 0 to 1. In this paper, we set CS as 0.9.

IT: Iteration Times, for each temperature, the SA must run IT times iteration to find a better result as the worst case. In this paper, we set IT as 100.

SIT: Stable Iteration Times, once the SA cannot find a better result for continuous SIT times at the particular temperature, the SA should reduce the temperature directly and search the result in the new round (at new temperature). In this paper, we set SIT as 50.

In this paper, we use the $Mutation$ function in GA as the Cs mapping refine function in the SA.

Algorithm 3 SA Algorithm Flow

SA starts:
Read Application (CTG) and NoCs Information;
Load SA parameters;
if $Temp > ETemp$ **then**
  while $k < IT$ **do**
    k = 0;
    Refine all the Cs mapping;
    Calculate the iteration cost value (I−cost);
    Generate the random probability P;
    if I−cost < H−cost or P < exp [(−1)*(I−cost − H−cost)/$Temp$] **then**
      Mapping accepted, H−cost = I−cost, m = 0, k++;
    else
      if $m < SIT$ **then**
        m++;
      else
        goto Cool $Temp$;
      end if
    end if
    Cool $Temp$, $Temp$ = CS * $Temp$;
  end while
end if
End: Output the $PCTG$ **(With** CTG **Allocated)**;

V. Experimental Results Analysis

We engaged the all experiments by using GSNOC Simulator with XHiNoC router [11], assuming the length of single-bit link (wire) on XY planar as 2.5 mm, the length of single-bit TSV is 20 μm, 1 GHz clock frequency, 25% bandwidth injection rate, with $SBSM$ [12] routing algorithm, differently in 8x8x4 and 4x4x4 network. All the physical power parameters are extract from SYNOPSYS using the TSMCs 65 nm technology.

Generic Scalable Pseudo Application (GSPA)

In our previous work [11], we have generated a random based generic scalable pseudo application (GSPA) test scenario, which can provide a complex and fully random task graph. The generated GSPA can provide designers the generic, scalable test scenarios for large systems. In this paper, we set the GSPA as 1000 tasks, maximal communication weight for each communication edge is 1000 flits, maximal execution time for each task is 100 clock cycles, and randomly generated the task dependencies.

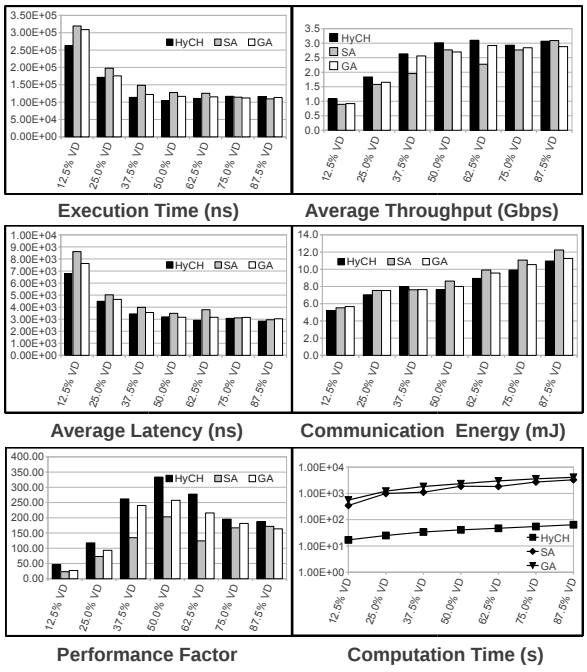

Fig. 3. GSPA Experimental Results

In the GSNOC platform, referring to Figure 1, we assume that at each network tile there is one piece of distributed memory (Local memory) and once the simulation started, we waited for a period of clock cycles to let the application information (TG) be booted into the system, and according to the traffic profile, the kernel would transmit/receive and execute the application data.

Figure 3 has shown the experimental results for the GSPA simulation. We set seven different VD as from 12.5% to 87.5% with symmetric VFs. In comparison to the GA method, our HyCH can save 5% overall execution time (Exe.Time), achieve 9% more average throughput (Avg.Tpt), save 5% and 4% average latency (Avg.Lat) and network communication energy (NCE), in average. In comparison to SA, our HyCH can save 11% Exe.Time, achieve 18% Avg.Tpt, save 12% and 7% Avg.Lat and NCE, in average.

We assumed the diameter of one single bit TSV pitch (d_{TSV}) as 5 μm, therefore the $Area_{TSV}$ can be directly calculated as

$$Area_{TSV} = VD \times X \times Y \times n_w \times d_{TSV}^2; \quad (9)$$

With different VD, the area of the NoCs systems are different. Here we model the area as

$$Area = Area_{R_{3D}} + Area_{R_{2D}} + Area_{TSV} + Area_{Wires} \quad (10)$$

$Area_{R_{3D}}$ and $Area_{R_{2D}}$ indicates the area of 3D and 2D NoCs routers, respectively. We synthesized all of the router components (in VHDL) and the interconnect wires using

TABLE I
NORMALIZED BENCHMARK EXPERIMENTAL RESULTS

Normalized	Exe.Time	Exe.Time	Avg.Tpt	Avg.Tpt	Avg.Lat	Avg.Lat	NCE	NCE	PF	PF
Application	HyCH/GA	HyCH/SA	HyCH/GA	HyCH/SA	HyCH/GA	HyCH/SA	HyCH/GA	HyCH/SA	HyCH/GA	HyCH/SA
blackscholes	1.00	1.00	1.00	1.00	1.02	1.02	1.16	0.91	1.09	1.12
bodytrack	0.99	0.95	0.96	1.08	1.01	0.94	1.03	1.01	0.94	1.20
fluidanimate	0.93	0.97	1.16	1.10	0.97	0.96	1.03	1.03	1.23	1.15
plsa	0.99	0.99	1.10	1.15	0.92	1.00	1.14	1.18	1.07	0.98
mpeg2enc	0.98	1.06	0.96	0.89	0.93	1.04	0.82	0.72	1.28	1.11
lu	0.93	0.91	1.10	1.13	0.96	0.91	1.02	1.10	1.19	1.23
Average	0.97	0.98	1.05	1.06	0.97	0.98	1.03	0.99	1.09	1.12

SYNOPSYS design compiler with TSMC 65 nm LP library and we obtained the first estimated area number.

We defined another concept to deliver the system performance factor (PF), which can be calculated as

$$PF = \frac{Avg.Tpt}{Exe.Time \times Avg.Lat \times NCE \times Area}; \qquad (11)$$

According to (11), we can easily evaluate the PF of the systems which employed HyCH, GA and SA. As shown on Figure 3, the HyCH can achieve in average 20.4% and 58% more PF value than the GA and SA method can, respectively, and the 50% VD with the certain setup achieved the most PF value and it matched the best trade-off point with all the constrains.

Regarding to the design time, we ran the HyCH, GA and SA on a Linux Debian 64 bit machine with Intel Xeon CPU (2.67G Hz), the HyCH calculation time is only 1% to 2% of the GA and SA calculation time (for example, with 87.5% VD and 256 $Tile$s 3D NoCs, HyCH cost 64 s, GA cost 4090 s and SA cost 3288 s).

We also took several applications from four benchmark suits (PARSEC [13], SPLASH−2 [14], ALPBench [15] and NU−MineBench [16]) for experiments in this subsection. We set the VD as 50% for all the benchmark experiments, and according to application constrains, we set the NoCs scale as 4x4x4 with 64 PEs. The experimental results have shown as TABLE I.

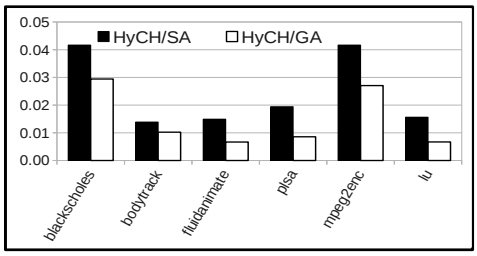

Fig. 4. Normalized Design Time Comparison between HyCH, GA and SA Methods against Different Benchmark Applications

VI. CONCLUSION

In this paper, we developed a Hybridize Constructive Heuristic (HyCH) method to allocate tasks on low vertical link density 3D NoCs based many core systems. In comparison to the classic SA and GA methods, our HyCH provides better performance, less communication overhead (in GSPA experiment, up to 23% and 15% less execution time, up to 36% and 18% higher average throughput, up to 21% and 11% lower average latency, up to 11% and 8% less network communication energy than SA and GA, respectively) and less design computation time.

REFERENCES

[1] L. Benini, G. De Micheli, "A New SoC Paradigm," *IEEE Computer*, pp. 70–78, January, 2005.

[2] G. Philip, B. Christopher, P. Ramm, *Handbook of 3D Integration: Technology and Applications of 3D Integrated Circuits*. Wiley-VCH, 2008.

[3] M. Motoyoshi, "Through-Silicon Via (TSV)," 2009.

[4] International Technology Roadmap for Semiconductors (ITRS), *www.itrs.net*.

[5] A. Hsieh, T. Hwang, M. Chang, M. Tsai, C. Tseng, H. Li, "TSV redanduncy: Architecture and design issues in 3D IC," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010.

[6] Y. Chen, L. Xie, J. Li, "An energy-aware heuristic constructive mapping algorithm for Network on Chip," *IEEE 8th International Conference on ASIC (ASICON)*, 2009.

[7] N. Choudhary, M. Gaur, V.Laxmi, V. Singh , "GA Based Congestion Aware Topology Generation for Application Specific NoC," *IEEE International Symposium on Electronic Design, Test and Application*, 2011.

[8] X. Jin, T. Watanabe , "An Efficient 3D NoC Synthesis by Using Genetic Algorithms ," *IEEE Region 10 Conference (TENCON)*, 2010.

[9] B. Hredzak, O. Diessel, "Optimization of placement of dynamic network-on-chip cores using simulated annealing," *37th Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2011.

[10] Y. Zhou, W. Sheng, X. Liu, W. He, Z. Mao, "Efficient temporal task partition for coarse-grain reconfigurable systems based on Simulated Annealing Genetic Algorithm," *IEEE 9th International Conference on ASIC (ASICON)*, 2011.

[11] H. Ying, A. Jaiswal, M. El-Ghany, T. Hollstein, K. Hofmann , "A Simulation Framework for 3-Dimension Networks-on-Chip with Different Vertical Channel Density Configurations ," *15th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2012.

[12] H. Ying, A. Jaiswal, K. Hofmann , "Deadlock-Free Routing Algorithms for 3-Dimension Networks-on-Chip with Reduced Vertical Channel Density Topologies ," *International Conference on High Performance Computing & Simulation, Workshop Dynamic Reconfigurable Network-on-Chip (HPCS-DRNoC)*, 2012.

[13] G. Bienia, *Benchmarking Modern Multiprocessors, PhD Thesis*. Princeton University, 2011.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, "The SPLASH-2 Programs: Characterizing and Methodological Considerations," *22nd International Symposium on Computer Architecture*, 1995.

[15] S. Adve, M. Li, R. Sasanka, Y. Chen, "The Alpbench Benchmark Suit for Complex Multimedia Applications," *IEEE International Symposium on Circuits and Systems*, 2010.

[16] W. Liao, J. Pisharath, Y. Liu, A. Choudhary, "Nu-MineBench 2.0," *Center for Ultra-Scale Computing and Information Security Technical Report, CUCIS-2005-08-01,*, 2005.